Decentralized Scheduling with Data Locality for Data-Parallel Computation on Peer-to-Peer Networks

Weina Wang, Matthew Barnard, and Lei Ying
School of Electrical, Computer and Energy Engineering
Arizona State University, Tempe, AZ 85287
{weina.wang, matthew.barnard, lei.ying.2}@asu.edu

Abstract—**Despite distributed in computation and data storage, current data-parallel computing systems are centralized in task scheduling, which results in hierarchies that create single point of failure, limit scalability, and increase administration costs. In this paper, we propose a fully decentralized scheduling algorithm for data-parallel computing systems on peer-to-peer (P2P) networks. Our scheduling algorithm eliminates the centralized scheduler by letting each node in the network make scheduling decisions. To achieve good performance, data locality, which stresses the efficiency of colocating tasks with their input data, and load-balancing, should be considered jointly, and in a decentralized fashion. By exploring a backpressure-based approach, the proposed task scheduling algorithm strikes the right balance between data locality and load-balancing with each node only knowing the status information of part of the nodes in the network, and proves to maximize the throughput.**

I. Introduction

Data-parallel computing systems, such as MapReduce [1] and Hadoop [2], are widely deployed in large computing clusters to meet the growing need for big data analysis [3]–[7]. Current implementations of these systems are usually specifically for high performance in centralized datacenter environments, using the master/slave architecture to coordinate the allocation of resources. As the development of volunteer computing systems that offer low-cost resource for high-performance computing, accommodating data-parallel computing systems to heterogeneous or opportunistic environments has attracted attention. However, despite of taking into account the possible unavailability of computation nodes, most of the existing work [8]–[11] still performs task scheduling on a reliable node, creating centralized control that is subject to the single point of failure. The work [12] implements MapReduce on a peer-to-peer (P2P) architecture, but it still uses the master/slave architecture for coordination, and there is a set of nodes dedicated to the role of masters.

In this paper, we propose to design a fully decentralized scheduling algorithm for data-parallel computing systems on P2P networks, where centralization is eliminated by distributing scheduling responsibility among nodes in the system. With such an approach, each node in the system will be not only a computation and data storage unit, as in MapReduce, but also a scheduling unit.

A data-parallel computing system typically divides large datasets into data chunks and stores them on nodes distributedly to cope with parallel computation [13], [14]. Each computation task that arrives at the system has a data chunk as its input. Therefore, when assigning tasks to nodes for execution, a critical consideration is to place tasks on or close to nodes that store the input data chunks to avoid data retrieval from other nodes. The concern regarding this problem is called *data locality* [15]–[24]. The data locality concern should be addressed with the load-balancing concern jointly in task scheduling, since improving data locality, i.e., assigning more tasks to nodes that have their input data, can leads to an uneven load distribution among nodes, and thus harms the throughput.

Therefore, to achieve good performance, the task scheduling algorithm needs to strike the right balance between data locality and load-balancing, and more importantly, do it *in a decentralized fashion*. Recently, centralized throughput optimal algorithms have been proposed for task scheduling with data locality concerns [21]–[24]. However, for a decentralized structure, we need to take into account the constraints on information exchange and cope with the lack of information. In a P2P network, a node only has access to the information of part of the nodes, so it needs to make scheduling decisions based on this limited knowledge, whereas a centralized scheduler knows the status of all nodes.

We address this difficulty by exploring a backpressure-based approach. Specifically, each node in the system maintains three queues on it. Two of them are called *execution queues*, which buffer tasks that will be executed on this node, and the other one is called the *forwarding queue*, which buffers tasks that will be delegated to other nodes. One of the execution queues is called the *local queue*, and the tasks in this queue have their input data on this node. The other execution queue is called the *remote queue*, and the tasks in this queue may need to retrieve their input data from other nodes. When a task arrives at the system, the node that receives the task first queries the P2P network to resolve

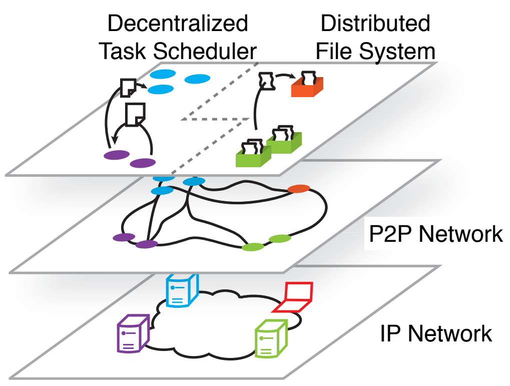

Fig. 1: The architecture of a data-parallel computing system on a P2P network.

the locations of its input data, and then assigns it to one of its local nodes according to their workloads of processing and forwarding. When choosing tasks to serve, a node computes the backpressure of the two execution queues as weighted queue lengths, where the weights correspond to the service rates such that tasks with data on the node has higher priority. When forwarding tasks in the network, a node computes the backpressure of the forwarding queue as the differences between it and the remote queues on its neighbors, and between it and the forwarding queues on its neighbors. Since the execution queues reflect nodes' working load, forwarding tasks along the decreasing direction of backpressure will lead tasks to a less loaded destination.

This proposed task scheduling algorithm does not require global information of all the nodes in the network. It assigns tasks preferably to their local nodes, but also makes use of the network to distribute some load to remote nodes, therefore fully utilizing the capacity. We prove that this proposed task scheduling algorithm maximizes the throughput.

II. System Model

We consider a data-parallel computing system on a P2P network with the architecture shown in Fig. 1. The three primary components in this computing system are a software P2P overlay network, a distributed file system (DFS), and a decentralized task scheduler. The distributed file system and task scheduler work together on all nodes; i.e., each node in the network supports the same service stack on data (e.g., storage, duplication, transmition) and tasks (e.g., execution, delegation).

Consider a computing system with N nodes and denote the set of nodes by $[N] = \{1, 2, \dots, N\}$. For each node n, we call the nodes that can be directly reached by n (i.e., the nodes whose IP addresses are known to node n) its *children*, or its *neighbors*, and correspondingly node n is called a *parent* of these nodes. Let $\mathcal{C}_n$ and $\mathcal{P}_n$ denote the sets of node n's children and parents, respectively. Data-parallel computing tasks come into the system stochastically

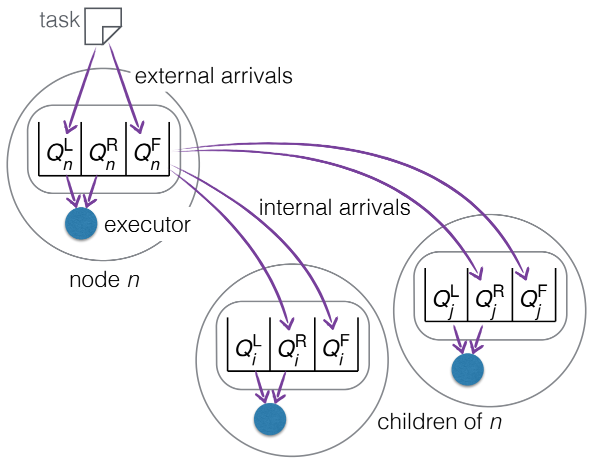

Fig. 2: Queue structure for task scheduling.

and need to be served by the nodes. We assume that each task is associated with a data chunk, and a data chunk is replicated and placed at several nodes by the P2P network. For each task, we call a node a *local node* for the task if the data chunk associated with the task is stored locally, and we call this task a *local task* on the node; otherwise, the node is called a *remote node* for the task and correspondingly this task is called a *remote task* on the node. Then each task is associated with a set of local nodes. We classify tasks into types according to their associated sets of local nodes, and let a type be denoted by this set of local nodes, $\mathcal{L} \subseteq [N]$. Thus a node n is a local node for tasks of type $\mathcal{L}$ if $n \in \mathcal{L}$.

A. Arrival and Service

We consider a time-slotted system. Let $A_{\mathcal{L}}(t)$ denote the number of tasks of type $\mathcal{L}$ arriving at the beginning of time slot t. We assume that the arrival process $(A_{\mathcal{L}}(t), t \geq 0)$ is temporally i.i.d. with arrival rate $\lambda_{\mathcal{L}}$, i.e., $\mathbb{E}[A_{\mathcal{L}}(t)] = \lambda_{\mathcal{L}}$ for any $t \geq 0$. We further assume that all the arrival processes are bounded. We only consider the types such that $\lambda_{\mathcal{L}} > 0$ to avoid triviality. Let $\lambda = (\lambda_{\mathcal{L}})$ denote the arrival rate vector.

The processing times of tasks are assumed to follow geometry distributions and the execution of tasks are nonpreemptive. Let α and γ denote the parameters of the geometry distributions for local tasks and remote tasks, respectively. Then the average service time is $1/\alpha$ for a local task, and it is $1/\gamma$ for a remote task. Due to data locality, we assume that $\gamma < \alpha$ to model the efficiency difference. We assume that the transmission of a task from one node to its children in the network is fast since the task consists of only some codes and meta data. Let the rates in the system be normalized such that the transmission rate is 1 task per time slot. Note that $\gamma < \alpha < 1$.

B. Queue Structure

We propose to use a queue structure as shown in Fig. 2 on each node to buffer tasks that wait to be executed or delegated. Each node n maintains three queues for tasks: a *local* queue Q_n^{L}, a *remote* queue Q_n^{R}, and a *forwarding* queue Q_n^{F}. The tasks in the local queue and remote queue will be executed on node n, and thus Q_n^{L} and Q_n^{R} are called *execution* queues. The local queue Q_n^{L} contains tasks whose input data is stored on n while the tasks in the remote queue Q_n^{R} may need to retrieve their input data from a remote node. The forwarding queue Q_n^{F} stores tasks that will be further delegated.

When a task arrives at the system, we call the node that receives the task the *initial node*. We assume that the task will be first sent to its local nodes by the initial node, and one of the local nodes will be selected to further handle the task. This assumption models the scenario that the distributed file system first locates the input data of a task according to the identifier specified in the task for further information or for some validation such as integrity check in practice. For example, in a distributed file system based on distributed hash table (DHT), the input data can be located through a key lookup process, and the local nodes are the responsible nodes for the queried key.

For each node n, the tasks that just arrive at the system and are assigned to node n as one of their local nodes by the initial nodes are called *external arrivals*, while the tasks that are delegated to node n by its parents are called *internal arrivals*. External arrivals are either accepted into Q_n^{L} for local execution or put into the forwarding queue Q_n^{F} for delegation. Internal arrivals are either accepted into Q_n^{R} for execution or put into the forwarding queue Q_n^{F} for further delegation.

III. Decentralized Scheduling with Data Locality

In this section, we present our scheduling algorithm based on the proposed queue structure. This scheduling algorithm is fully decentralized in the sense that all the nodes are equal in making scheduling decisions.

A. Task Scheduling Algorithm

A centralized scheduler has the benefit of intelligently allocating load given its global knowledge of the status of all the nodes. Contrastingly, a node in a P2P network has only partial knowledge of the nodes that it has connections with. Our scheduling algorithm overcomes this difficulty by using a backpressure-based method, where the properly built pressure pushes tasks in the network and leads them to an efficient destination.

The scheduler on each node operates by assigning newly arriving tasks to one of a set of execution and forwarding queues. Tasks in execution queues are scheduled for processing, while the forwarding queue diffuses tasks to the children. Data locality is taken into account by giving local tasks higher priority when making execution decisions, and routing tasks towards the decreasing direction of pressure balances loads on the nodes.

Formally, we propose the following backpressure scheduling algorithm, which is motivated by the backpressure algorithm for wireless networks [25], and is the multihop version of the task scheduling algorithm developed in [22].

Task Scheduling Algorithm:

- When a task of type $\mathcal{L}$ arrives at the system, the initial node assigns it to the shortest queue in $\{Q_n^{\mathrm{L}}, n \in \mathcal{L}\} \cup \{Q_n^{\mathrm{F}}, n \in \mathcal{L}\}$.
- For each node n, we consider the connections between the forwarding queue Q_n^{F} and the queues in $\{Q_m^{\mathrm{R}}, m \in \mathcal{C}_n\} \cup \{Q_m^{\mathrm{F}}, m \in \mathcal{C}_n\} \cup \{Q_n^{\mathrm{F}}\}$ as logical links. Define the backpressure of these links at time slot t as
$$w\big(Q_n^{\mathrm{F}}(t), Q_m^{\mathrm{R}}(t)\big) = Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{R}}(t), m \in \mathcal{C}_n, \quad (1)$$
$$w\big(Q_n^{\mathrm{F}}(t), Q_m^{\mathrm{F}}(t)\big) = Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{F}}(t), m \in \mathcal{C}_n, \quad (2)$$
$$w\big(Q_n^{\mathrm{F}}(t), Q_n^{\mathrm{F}}(t)\big) = 0. \quad (3)$$
Then one task in Q_n^{F} (if there is any) is forwarded to the queue in $\{Q_m^{\mathrm{R}}, m \in \mathcal{C}_n\} \cup \{Q_m^{\mathrm{F}}, m \in \mathcal{C}_n\} \cup \{Q_n^{\mathrm{F}}\}$ with the maximum backpressure at each time slot. Note that if this queue is Q_n^{F}, then it means no task will be forwarded out from Q_n^{F}.
- For each node n, we consider the connections between the queues in $\{Q_n^{\mathrm{L}}, Q_n^{\mathrm{R}}\}$ and the processor of node n as logical links. Define the backpressure of these links at time slot t as
$$w\big(Q_n^{\mathrm{L}}(t), n\big) = \alpha Q_n^{\mathrm{L}}(t),\ \ w\big(Q_n^{\mathrm{R}}(t), n\big) = \gamma Q_n^{\mathrm{R}}(t), \quad (4)$$
where recall that $1/\alpha$ is the average service time for a local task and $1/\gamma$ is the average service time for a remote task. If a node just finished a task at time slot $t-1$, we say the node is *available* at time slot t, and otherwise the node is *busy* since it must be serving some task. Then at each time slot, if node n is available, it is scheduled to serve a task from the queue in $\{Q_n^{\mathrm{L}}, Q_n^{\mathrm{R}}\}$ with maximum backpressure; otherwise node n continues serving its unfinished task, i.e., the execution of tasks is *nonpreemptive*.

B. Queue Dynamics

At the beginning of time slot t, the external and internal arrivals are assigned to queues according to the task scheduling algorithm. For each type $\mathcal{L}$, external arrivals of this type are assigned to either the local queues or the forwarding queues of their local nodes. We denote the number of external arrivals allocated to Q_n^{L} and Q_n^{F} by $A_{\mathcal{L},n}^{\mathrm{L}}(t)$ and $A_{\mathcal{L},n}^{\mathrm{FE}}(t)$, respectively, where $n \in \mathcal{L}$. Then $A_{\mathcal{L}}(t) = A_{\mathcal{L},n}^{\mathrm{L}}(t) + A_{\mathcal{L},n}^{\mathrm{FE}}(t)$. Let $A_n^{\mathrm{L}}(t)$ denote the total number of arrivals to Q_n^{L}, and $A_n^{\mathrm{FE}}(t)$ denote the total number of external arrivals to Q_n^{F}. Then $A_n^{\mathrm{L}}(t) = \sum_{\mathcal{L}} A_{\mathcal{L},n}^{\mathrm{L}}(t)$ and $A_n^{\mathrm{FE}}(t) = \sum_{\mathcal{L}} A_{\mathcal{L},n}^{\mathrm{FE}}(t)$.

Let $A_{kn}^{\mathrm{R}}(t)$ and $A_{kn}^{\mathrm{FI}}(t)$ denote the number of internal arrivals that are forwarded to Q_n^{R} and Q_n^{F} from a parent node k, respectively. Then $A_n^{\mathrm{R}}(t) = \sum_{k \in \mathcal{P}_n} A_{kn}^{\mathrm{R}}(t)$ and

$A_n^{\mathrm{FI}}(t) = \sum_{k \in \mathcal{P}_n} A_{kn}^{\mathrm{FI}}(t)$ denote the total number of internal arrivals to Q_n^{R} and Q_n^{F}, respectively.

If node n just completed a task at time slot $t-1$, we say the node is *available* at time slot t, and otherwise the node is *busy* since it must be serving some task. The working status of node n is kept track of by $f_n(t)$, where $f_n(t) = 0, 1, 2$ denotes available, working on a local task, and working on a remote task, respectively. If node n is available and is scheduled to serve the local queue, or if the node is busy serving the local queue, we say the node is scheduled to the local queue. Otherwise we say the node is scheduled to the remote queue. The scheduling decision of node n is recorded by

$$\sigma_n(t) = \begin{cases} 1 & \text{if node } n \text{ is scheduled to the local queue,} \\ 2 & \text{if node } n \text{ is scheduled to the remote queue.} \end{cases} \tag{5}$$

At the end of time slot t, finished tasks depart and the queue lengths and the working statuses are updated. Define

$$\mu_n^{\mathrm{L}}(t) = \begin{cases} \alpha & \text{if } \sigma_n(t) = 1, \\ 0 & \text{if } \sigma_n(t) = 2, \end{cases} \quad \mu_n^{\mathrm{R}}(t) = \begin{cases} 0 & \text{if } \sigma_n(t) = 1, \\ \gamma & \text{if } \sigma_n(t) = 2. \end{cases} \tag{6}$$

Let random variables $S_n^{\mathrm{L}}(t)$ and $S_n^{\mathrm{R}}(t)$ follow Bernoulli distributions with parameter μ_n^{L} and μ_n^{R}, respectively. Then $S_n^{\mathrm{L}}(t)$ and $S_n^{\mathrm{R}}(t)$ represent the service performed by node n to the local queue Q_n^{L} and Q_n^{R}, respectively. Let $S_n^{\mathrm{F}}(t)$ denote the number of departures of the forwarding queue Q_n^{F}. Recall that we normalize the rates in the system such that the forwarding rate is 1. Then $S_n^{\mathrm{F}}(t)$ is either 1 or 0. By the above notation, the queue dynamics can be written as

$$Q_n^{\mathrm{L}}(t+1) = (Q_n^{\mathrm{L}}(t) + A_n^{\mathrm{L}}(t) - S_n^{\mathrm{L}}(t))^+, \tag{7}$$
$$Q_n^{\mathrm{R}}(t+1) = (Q_n^{\mathrm{R}}(t) + A_n^{\mathrm{R}}(t) - S_n^{\mathrm{R}}(t))^+, \tag{8}$$
$$Q_n^{\mathrm{F}}(t+1) = (Q_n^{\mathrm{F}}(t) + A_n^{\mathrm{FE}}(t) + A_n^{\mathrm{FI}}(t) - S_n^{\mathrm{F}}(t))^+. \tag{9}$$

We assemble the queue lengths and working statuses into a vector:

$$Z(t) = (Q_n^{\mathrm{L}}(t), Q_n^{\mathrm{R}}(t), Q_n^{\mathrm{F}}(t), f_n(t), n \in [N]). \tag{10}$$

Then $\{Z(t), t \geq 0\}$ is a Markov chain. We assume that the state space $\mathcal{S}$ consists of all the states which can be reached from the zero vector.

IV. Throughput Optimality

In this section, we analyze the throughput performance of our task scheduling algorithm. We first derive an outer bound of the capacity region of the system, and then show that our task scheduling algorithm is throughput optimal.

For any time slot t, let $\Phi(t)$ be the number of unfinished tasks in the system, i.e., $\Phi(t) = \sum_{n=1}^{N} \big(Q_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t) + Q_n^{\mathrm{F}}(t)\big)$. We say that a task scheduling algorithm *stabilizes* the system for an arrival rate vector $\lambda = (\lambda_{\mathcal{L}})$ if

$$\lim_{C \to \infty} \lim_{t \to \infty} \Pr(\Phi(t) > C) = 0. \tag{11}$$

Intuitively, stability indicates that the number of unfinished tasks will not explode.

The capacity region of a system is the set of arrival rate vectors for which there exists a task scheduling algorithm that stabilizes the system. A task scheduling algorithm is throughput optimal if it stabilizes the system for any arrival rate vector strictly within the capacity region.

A. Capacity Region

We first characterize the capacity region of the system by considering some necessary conditions for an arrival rate vector $\lambda = (\lambda_{\mathcal{L}})$ to be in the capacity region.

Consider the following set of arrival rate vectors:

$$\Lambda = \Bigg\{ \lambda = (\lambda_{\mathcal{L}})\colon \exists \lambda_{\mathcal{L},n}^{\mathrm{L}}, \lambda_{\mathcal{L},n}^{\mathrm{F}}, \lambda_{nm}^{\mathrm{R}}, \lambda_{nm}^{\mathrm{F}}, \lambda_n^{\mathrm{L}}, \lambda_n^{\mathrm{F}}, \lambda_n^{\mathrm{R}}, \text{s.t.}$$
$$\lambda_{\mathcal{L}} = \sum_{n \in \mathcal{L}} \big(\lambda_{\mathcal{L},n}^{\mathrm{L}} + \lambda_{\mathcal{L},n}^{\mathrm{F}}\big), \forall \mathcal{L},$$
$$\lambda_n^{\mathrm{L}} = \sum_{\mathcal{L}\colon n \in \mathcal{L}} \lambda_{\mathcal{L},n}^{\mathrm{L}}, \; \lambda_n^{\mathrm{F}} = \sum_{\mathcal{L}\colon n \in \mathcal{L}} \lambda_{\mathcal{L},n}^{\mathrm{F}}, \forall n \in [N],$$
$$\lambda_n^{\mathrm{F}} + \sum_{k \in \mathcal{P}_n} \lambda_{kn}^{\mathrm{F}} = \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{R}} + \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{F}}, \forall n \in [N],$$
$$\lambda_n^{\mathrm{R}} = \sum_{k \in \mathcal{P}_n} \lambda_{kn}^{\mathrm{R}}, \forall n \in [N],$$
$$\lambda_{\mathcal{L},n}^{\mathrm{L}}, \lambda_{\mathcal{L},n}^{\mathrm{F}}, \lambda_{nm}^{\mathrm{R}}, \lambda_{nm}^{\mathrm{F}} \geq 0, \forall \mathcal{L}, \forall n \in [N], \forall m \in \mathcal{C}_n,$$
$$\frac{\lambda_n^{\mathrm{L}}}{\alpha} + \frac{\lambda_n^{\mathrm{R}}}{\gamma} \leq 1, \forall n \in [N],$$
$$\sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{R}} + \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{F}} \leq 1, \forall n \in [N] \Bigg\}.$$

Then it can be proved that no task scheduling algorithm can stabilize the system for an arrival rate vector that is not in Λ. In other words, if the system can be stabilized by some task scheduling algorithm, then the arrival rate vector $\lambda = (\lambda_{\mathcal{L}})$ must be in Λ. Therefore, Λ gives an outer bound of the capacity region of the system. The proof is similar to the proof of Theorem 5.3.1 in [26], which is based on the strict separation theorem and strong law of large numbers. We omit the proof here for conciseness.

Intuitively, we can think of these elements in the definition of Λ as the following rates:

- $\lambda_{\mathcal{L},n}^{\mathrm{L}}$: rate of the tasks of type $\mathcal{L}$ that are *finally* executed on a local node $n \in \mathcal{L}$;
- $\lambda_{\mathcal{L},n}^{\mathrm{F}}$: rate of the tasks of type $\mathcal{L}$ that are *finally* executed on a remote node and are initially forwarded out from the local node $n \in \mathcal{L}$;
- $\lambda_{nm}^{\mathrm{R}}$: rate of the tasks that are forwarded from node n to its child m for remote execution;
- $\lambda_{nm}^{\mathrm{R}}$: rate of the tasks that are forwarded from node n to its child m for further delegation.

Then the last two inequalities in the definition of Λ represent the capacity constraints of processing and forwarding.

B. Achievability

Theorem 1. *The proposed backpressure scheduling algorithm for task scheduling stabilizes the system for any arrival rate vector strictly within* Λ.

By this theorem, the proposed task scheduling algorithm is throughput optimal. For any arrival rate vector, as long as there exists some task scheduling algorithm that stabilizes the system under the considered queue structure, the proposed algorithm can stabilize the system for this arrival rate vector.

For any arrival rate vector $\lambda \in \Lambda^o$, by the assumptions and the proposed scheduling algorithm, $\{Z(t), t \geq 0\}$ is an irreducible and aperiodic Markov chain, and the number of unfinished tasks $\Phi(t) = \sum_{m=1}^{M+1} Q_m(t)$. If this Markov chain is positive recurrent, then the condition of stability is satisfied since the distribution of $Z(t)$ will converge to the stationary distribution as $t \to \infty$. Thus the stability can be obtained by proving the positive recurrence. A difficulty of the proof is that the execution of tasks is nonpreemptive. We use techniques developed in [21], [22] to address this difficulty.

Proof of Theorem 1: Consider the following Lyapunov function

$$V(Z(t)) = \sum_{n=1}^{N} \big((Q_n^{\mathrm{L}}(t))^2 + (Q_n^{\mathrm{R}}(t))^2 + (Q_n^{\mathrm{F}}(t))^2\big). \quad (12)$$

Then according to an extension of the Foster-Lyapunov theorem [26], we need to prove that there exists a finite set $\mathcal{B} \subseteq \mathcal{S}$ and two constants δ and C with $\delta > 0$ such that for some positive integer T and any $t_0 \geq 0$,

$$\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0) = Z\big] \leq -\delta, \forall Z \in \mathcal{B}^c,$$
$$\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0) = Z\big] \leq C, \ \forall Z \in \mathcal{B}.$$

For any arrival rate vector $\lambda \in \Lambda^o$, we show that the queueing process $\{Z(t), t \geq 0\}$ satisfies these two conditions.

Consider any arrival rate vector $\lambda \in \Lambda^o$. Since Λ^o is an open set, there exists $\epsilon > 0$ such that $\lambda' = (1+\epsilon)\lambda \in \Lambda$. Therefore $\lambda = (\lambda_{\mathcal{L}})$ has a decomposition such that

$$\lambda_{\mathcal{L}} = \sum_{n \in \mathcal{L}} \big(\lambda_{\mathcal{L},n}^{\mathrm{L}} + \lambda_{\mathcal{L},n}^{\mathrm{F}}\big), \forall \mathcal{L}, \quad (13)$$

$$\lambda_n^{\mathrm{L}} = \sum_{\mathcal{L}:\, n \in \mathcal{L}} \lambda_{\mathcal{L},n}^{\mathrm{L}},\ \lambda_n^{\mathrm{F}} = \sum_{\mathcal{L}:\, n \in \mathcal{L}} \lambda_{\mathcal{L},n}^{\mathrm{F}}, \forall n \in [N], \quad (14)$$

$$\lambda_n^{\mathrm{F}} + \sum_{k \in \mathcal{P}_n} \lambda_{kn}^{\mathrm{F}} = \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{R}} + \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{F}}, \forall n \in [N], \quad (15)$$

$$\lambda_n^{\mathrm{R}} = \sum_{k \in \mathcal{P}_n} \lambda_{kn}^{\mathrm{R}}, \forall n \in [N], \quad (16)$$

$$\lambda_{\mathcal{L},n}^{\mathrm{L}}, \lambda_{\mathcal{L},n}^{\mathrm{F}}, \lambda_{nm}^{\mathrm{R}}, \lambda_{nm}^{\mathrm{F}} \geq 0, \forall \mathcal{L}, \forall n \in [N], \forall m \in \mathcal{C}_n, \quad (17)$$

$$\frac{\lambda_n^{\mathrm{L}}}{\alpha} + \frac{\lambda_n^{\mathrm{R}}}{\gamma} \leq \frac{1}{1+\epsilon}, \forall n \in [N], \quad (18)$$

$$\sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{R}} + \sum_{m \in \mathcal{C}_n} \lambda_{nm}^{\mathrm{F}} \leq \frac{1}{1+\epsilon}, \forall n \in [N]. \quad (19)$$

By the queue dynamics, the T time slot Lyapunov drift can be calculated as

$$\begin{aligned}
&\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0)\big] \\
&= \mathbb{E}\Bigg[\sum_{t=t_0}^{t_0+T-1} \sum_{n=1}^{N} \Big((Q_n^{\mathrm{L}}(t+1))^2 - (Q_n^{\mathrm{L}}(t))^2 \\
&\qquad + (Q_n^{\mathrm{R}}(t+1))^2 - (Q_n^{\mathrm{R}}(t))^2 \\
&\qquad + (Q_n^{\mathrm{F}}(t+1))^2 - (Q_n^{\mathrm{F}}(t))^2 \Big) \Bigm| Z(t_0)\Bigg] \\
&\leq 2\mathbb{E}\Bigg[\sum_t \sum_n \Big(Q_n^{\mathrm{L}}(t)\big(A_n^{\mathrm{L}}(t) - S_n^{\mathrm{L}}(t)\big) \\
&\qquad + Q_n^{\mathrm{R}}(t)\big(A_n^{\mathrm{R}}(t) - S_n^{\mathrm{R}}(t)\big) \\
&\qquad + Q_n^{\mathrm{F}}(t)\big(A_n^{\mathrm{FE}}(t) + A_n^{\mathrm{FI}}(t) - S_n^{\mathrm{F}}(t)\big) \Big) \Bigm| Z(t_0)\Bigg] \\
&\quad + C_1,
\end{aligned}$$

where C_1 is a constant independent of $Z(t_0)$ since the arrival and service processes are bounded. We omit the summation ranges when they are clear in context. Rearranging the terms we get

$$\begin{aligned}
&\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0)\big] \\
&\leq 2\mathbb{E}\Bigg[\sum_t \sum_n \Big(Q_n^{\mathrm{L}}(t) A_n^{\mathrm{L}}(t) + Q_n^{\mathrm{F}}(t) A_n^{\mathrm{FE}}(t) \Big) \Bigm| Z(t_0)\Bigg] \quad (20) \\
&\quad + 2\mathbb{E}\Bigg[\sum_t \sum_n \Big(Q_n^{\mathrm{R}}(t) A_n^{\mathrm{R}}(t) + Q_n^{\mathrm{F}}(t) A_n^{\mathrm{FI}}(t) \\
&\qquad - Q_n^{\mathrm{F}}(t) S_n^{\mathrm{F}} \Big) \Bigm| Z(t_0)\Bigg] \quad (21) \\
&\quad - 2\mathbb{E}\Bigg[\sum_t \sum_n \Big(Q_n^{\mathrm{L}}(t) S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t) S_n^{\mathrm{R}}(t) \Big) \Bigm| Z(t_0)\Bigg] \quad (22) \\
&\quad + C_1.
\end{aligned}$$

We bound these three terms in the remainder of this proof.

First consider the term (20). For any time slot t with $t_0 \leq t \leq t_0 + T - 1$,

$$\begin{aligned}
&\mathbb{E}\Bigg[\sum_n \Big(Q_n^{\mathrm{L}}(t) A_n^{\mathrm{L}}(t) + Q_n^{\mathrm{F}}(t) A_n^{\mathrm{FE}}(t) \Big) \Bigm| Z(t_0)\Bigg] \\
&= \mathbb{E}\Bigg[\mathbb{E}\Bigg[\sum_n \Big(Q_n^{\mathrm{L}}(t) A_n^{\mathrm{L}}(t) \\
&\qquad + Q_n^{\mathrm{F}}(t) A_n^{\mathrm{FE}}(t) \Big) \Bigm| Z(t)\Bigg] \Bigm| Z(t_0)\Bigg] \\
&= \mathbb{E}\Bigg[\mathbb{E}\Bigg[\sum_n \sum_{\mathcal{L}:\, n \in \mathcal{L}} \Big(Q_n^{\mathrm{L}}(t) A_{\mathcal{L},n}^{\mathrm{L}}(t) \\
&\qquad + Q_n^{\mathrm{F}}(t) A_{\mathcal{L},n}^{\mathrm{FE}}(t) \Big) \Bigm| Z(t)\Bigg] \Bigm| Z(t_0)\Bigg]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}\Bigg[\mathbb{E}\Big[\sum_{\mathcal{L}}\sum_{n\in\mathcal{L}}\Big(Q_n^{\mathrm{L}}(t)A_{\mathcal{L},n}^{\mathrm{L}}(t) \\
&\qquad + Q_n^{\mathrm{F}}(t)A_{\mathcal{L},n}^{\mathrm{FE}}(t)\Big)\,\Big|\,Z(t)\Big]\,\Bigg|\,Z(t_0)\Bigg] \\
&\overset{(a)}{=} \mathbb{E}\Big[\sum_{\mathcal{L}} Q_{\mathcal{L}}^{*}(t)\mathbb{E}\big[A_{\mathcal{L}}(t)\,\big|\,Z(t)\big]\,\Big|\,Z(t_0)\Big] \\
&= \mathbb{E}\Big[\sum_{\mathcal{L}} Q_{\mathcal{L}}^{*}(t)\lambda_{\mathcal{L}}\,\Big|\,Z(t_0)\Big] \\
&\overset{(b)}{\le} \sum_{\mathcal{L}}\sum_{n\in\mathcal{L}}\Big(Q_n^{\mathrm{L}}(t_0)\lambda_{\mathcal{L},n}^{\mathrm{L}} + Q_n^{\mathrm{F}}(t_0)\lambda_{\mathcal{L},n}^{\mathrm{F}}\Big) + C_2, \\
&= \sum_{n}\Big(Q_n^{\mathrm{L}}(t_0)\lambda_n^{\mathrm{L}} + Q_n^{\mathrm{F}}(t_0)\lambda_n^{\mathrm{F}}\Big) + C_2, \qquad (23)
\end{aligned}
$$

where (a) follows from the scheduling algorithm with

$$Q_{\mathcal{L}}^{*}(t) = \min\Big\{\{Q_n^{\mathrm{L}}(t), n\in\mathcal{L}\}\cup\{Q_n^{\mathrm{F}}(t), n\in\mathcal{L}\}\Big\},$$

(b) follows from the decomposition (13), and C_2 is a constant independent of $Z(t_0)$.

Next consider the term (21). It can be shown that

$$
\begin{aligned}
&\sum_{t}\Big(Q_n^{\mathrm{R}}(t)A_n^{\mathrm{R}}(t) + Q_n^{\mathrm{F}}(t)A_n^{\mathrm{FI}}(t)\Big) \\
&\le \sum_{t}\Big(Q_n^{\mathrm{R}}(t)A_n^{\mathrm{R}}(t+1) + Q_n^{\mathrm{F}}(t)A_n^{\mathrm{FI}}(t+1)\Big) \\
&\quad + NQ_n^{\mathrm{R}}(t_0) + NQ_n^{\mathrm{F}}(t_0) + C_3,
\end{aligned}
$$

where C_3 is a constant independent of $Z(t_0)$. For any time slot t with $t_0 \le t \le t_0+T-1$,

$$
\begin{aligned}
\sum_n Q_n^{\mathrm{F}}(t)A_n^{\mathrm{FI}}(t+1) &= \sum_n Q_n^{\mathrm{F}}(t)\sum_{k\in\mathcal{P}_n} A_{kn}^{\mathrm{FI}}(t+1) \\
&= \sum_k\sum_{n\in\mathcal{C}_k} Q_n^{\mathrm{F}}(t)A_{kn}^{\mathrm{FI}}(t+1) \\
&\overset{(a)}{=} \sum_n\sum_{m\in\mathcal{C}_n} Q_m^{\mathrm{F}}(t)A_{nm}^{\mathrm{FI}}(t+1),
\end{aligned}
$$

where (a) is obtained by changing the names of the summation indices. Similarly,

$$\sum_n Q_n^{\mathrm{R}}(t)A_n^{\mathrm{R}}(t+1) = \sum_n\sum_{m\in\mathcal{C}_n} Q_m^{\mathrm{R}}(t)A_{nm}^{\mathrm{R}}(t+1).$$

For each node n, let

$$
\begin{aligned}
\Delta^{*}Q_n(t) = \max\Big\{&\{Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{R}}(t), m\in\mathcal{C}_n\} \\
&\cup\{Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{F}}(t), m\in\mathcal{C}_n\} \\
&\cup\{0\}\Big\}.
\end{aligned}
$$

By the scheduling algorithm,

$$\sum_{m\in\mathcal{C}_n} A_{nm}^{\mathrm{R}}(t+1) + \sum_{m\in\mathcal{C}_n} A_{nm}^{\mathrm{FI}}(t+1) = S_n^{\mathrm{F}}(t).$$

Therefore,

$$
\begin{aligned}
&\sum_{m\in\mathcal{C}_n} Q_m^{\mathrm{R}}(t)A_{nm}^{\mathrm{R}}(t+1) + \sum_{m\in\mathcal{C}_n} Q_m^{\mathrm{F}}(t)A_{nm}^{\mathrm{FI}}(t+1) \\
&\quad - Q_n^{\mathrm{F}}(t)S_n^{\mathrm{F}}(t) \\
&= \sum_{m\in\mathcal{C}_n}\big(Q_m^{\mathrm{R}}(t) - Q_n^{\mathrm{F}}(t)\big)A_{nm}^{\mathrm{R}}(t+1) \\
&\quad + \sum_{m\in\mathcal{C}_n}\big(Q_m^{\mathrm{F}}(t) - Q_n^{\mathrm{F}}(t)\big)A_{nm}^{\mathrm{FI}}(t+1) \\
&\overset{(a)}{=} -\Delta^{*}Q_n(t) \\
&= -\frac{\epsilon}{1+\epsilon}\Delta^{*}Q_n(t) - \frac{1}{1+\epsilon}\Delta^{*}Q_n(t) \\
&\overset{(b)}{\le} -\frac{\epsilon}{1+\epsilon}\Delta^{*}Q_n(t) - \Delta^{*}Q_n(t)\sum_{m\in\mathcal{C}_n}\big(\lambda_{nm}^{\mathrm{R}} + \lambda_{nm}^{\mathrm{F}}\big),
\end{aligned}
$$

where (a) follows from the scheduling algorithm, and (b) follows from the constraint (19). By the definition of $\Delta^{*}Q_n(t)$,

$$
\begin{aligned}
&-\sum_n \Delta^{*}Q_n(t)\sum_{m\in\mathcal{C}_n}\big(\lambda_{nm}^{\mathrm{R}} + \lambda_{nm}^{\mathrm{F}}\big) \\
&\le -\sum_n\sum_{m\in\mathcal{C}_n}\big(Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{R}}(t)\big)\lambda_{nm}^{\mathrm{R}} \\
&\quad -\sum_n\sum_{m\in\mathcal{C}_n}\big(Q_n^{\mathrm{F}}(t) - Q_m^{\mathrm{F}}(t)\big)\lambda_{nm}^{\mathrm{F}} \\
&= -\sum_n\sum_{m\in\mathcal{C}_n} Q_n^{\mathrm{F}}(t)\big(\lambda_{nm}^{\mathrm{R}} + \lambda_{nm}^{\mathrm{F}}\big) \\
&\quad +\sum_n\sum_{k\in\mathcal{P}_n}\big(Q_n^{\mathrm{R}}(t)\lambda_{kn}^{\mathrm{R}} + Q_n^{\mathrm{F}}(t)\lambda_{kn}^{\mathrm{F}}\big) \\
&\overset{(a)}{=} \sum_n Q_n^{\mathrm{R}}(t)\sum_{k\in\mathcal{P}_n}\lambda_{kn}^{\mathrm{R}} - \sum_n Q_n^{\mathrm{F}}(t)\lambda_n^{\mathrm{F}} \\
&= \sum_n Q_n^{\mathrm{R}}(t)\lambda_n^{\mathrm{R}} - \sum_n Q_n^{\mathrm{F}}(t)\lambda_n^{\mathrm{F}},
\end{aligned}
$$

where (a) follows from the decomposition (15). Then

$$
\begin{aligned}
&\mathbb{E}\Big[\sum_t\sum_n\Big(Q_n^{\mathrm{R}}(t)A_n^{\mathrm{R}}(t) + Q_n^{\mathrm{F}}(t)A_n^{\mathrm{FI}}(t) \\
&\qquad - Q_n^{\mathrm{F}}(t)S_n^{\mathrm{F}}(t)\Big)\,\Big|\,Z(t_0)\Big] \\
&= \mathbb{E}\Bigg[\mathbb{E}\Big[\sum_t\sum_n\Big(Q_n^{\mathrm{R}}(t)A_n^{\mathrm{R}}(t) + Q_n^{\mathrm{F}}(t)A_n^{\mathrm{FI}}(t) \\
&\qquad - Q_n^{\mathrm{F}}(t)S_n^{\mathrm{F}}(t)\Big)\,\Big|\,Z(t)\Big]\,\Bigg|\,Z(t_0)\Bigg] \\
&\le -\frac{\epsilon T}{1+\epsilon}\Delta^{*}Q_n(t_0) + T\sum_n Q_n^{\mathrm{R}}(t_0)\lambda_n^{\mathrm{R}} \\
&\quad - T\sum_n Q_n^{\mathrm{F}}(t_0)\lambda_n^{\mathrm{F}} + N\sum_n Q_n^{\mathrm{R}}(t_0) + N\sum_n Q_n^{\mathrm{F}}(t_0) \\
&\quad + C_4, \qquad (24)
\end{aligned}
$$

where C_4 is a constant independent of $Z(t_0)$.

Now consider the last term (22). Consider the following random variables

$$\begin{aligned} t_n^* &= \min\{\tau \colon \tau \geq t_0, f_n(\tau) = 0\}, \;\; n \in [N], \\ t^* &= \max_{1 \leq n \leq N} t_n^*. \end{aligned} \tag{25}$$

Then t_n^* is the first time slot after t_0 at which node n is available. We can use t^* to decompose the probability space. Let $T = JK$, where J and K are integers. Denote $P_K = \Pr\big(t^* < t_0 + K \mid Z(t_0)\big)$. Then for each node n,

$$\begin{aligned} &\mathbb{E}\Bigg[\sum_t \bigg(Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t)\bigg)\bigg| Z(t_0)\Bigg] \\ &\geq \mathbb{E}\Bigg[\sum_t \bigg(Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t)\bigg)\bigg| Z(t_0), t^* < t_0 + K\Bigg] \\ &\quad \cdot \Pr\big(t^* < t_0 + K \mid Z(t_0)\big) \\ &\geq \mathbb{E}\Bigg[\sum_{t=t^*}^{t_0+T-1} \mathbb{E}\bigg[Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t) \;\bigg|\; Z(t_0), t^*\bigg] \\ &\qquad\qquad \bigg|\; Z(t_0), t^* < t_0 + K\Bigg] \cdot P_K. \end{aligned}$$

Then

$$\begin{aligned} &\mathbb{E}\bigg[Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t) \;\bigg|\; Z(t_0), t^*\bigg] \\ &= \mathbb{E}\bigg[\mathbb{E}\Big[Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t) \;\Big|\; Z(t), Z(t_0), t^*\Big] \\ &\qquad \bigg|\; Z(t_0), t^*\bigg]. \end{aligned}$$

For any time slot t with $t^* \leq t \leq t_0 + T - 1$, let

$$\tau_n^t = \max\{\tau \colon \tau \leq t, f_n(\tau) = 0\}. \tag{26}$$

Then

$$\begin{aligned} &\mathbb{E}\Big[Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t) \;\Big|\; Z(t), Z(t_0), t^*\Big] \\ &= \mathbb{E}\Big[Q_n^{\mathrm{L}}(\tau_n^t)\mathbb{E}\big[S_n^{\mathrm{L}}(t) \mid \sigma(\tau_n^t)\big] \\ &\qquad + Q_n^{\mathrm{R}}(\tau_n^t)\mathbb{E}\big[S_n^{\mathrm{R}}(t) \mid \sigma(\tau_n^t)\big] \;\Big|\; Z(t), Z(t_0), t^*\Big] - C_5 \\ &\overset{(a)}{=} \mathbb{E}\Big[\max\big\{\alpha Q_n^{\mathrm{L}}(\tau_n^t), \gamma Q_n^{\mathrm{R}}(\tau_n^t)\big\} \;\Big|\; Z(t), Z(t_0), t^*\Big] - C_5 \\ &\geq \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} - C_6, \end{aligned}$$

where (a) follows from the scheduling algorithm, and C_5, C_6 are constants independent of $Z(t_0)$. Therefore,

$$\begin{aligned} &\mathbb{E}\Bigg[\sum_t \bigg(Q_n^{\mathrm{L}}(t)S_n^{\mathrm{L}}(t) + Q_n^{\mathrm{R}}(t)S_n^{\mathrm{R}}(t)\bigg) \;\bigg|\; Z(t_0)\Bigg] \\ &\geq \frac{J-1}{J} P_K T \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} - C_7, \end{aligned} \tag{27}$$

where C_7 is a constant independent of $Z(t_0)$. Since

$$\lim_{K \to \infty} P_K = 1,$$

we can pick large enough K and J such that

$$\frac{J-1}{J} P \geq 1 - \frac{\epsilon}{2(1+\epsilon)}.$$

Combining the bounds (23), (24), (27) on (20), (21), (22) yields

$$\begin{aligned} &\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \mid Z(t_0)\big] \\ &\leq 2T \sum_n \bigg(Q_n^{\mathrm{L}}(t_0)\lambda_n^{\mathrm{L}} + Q_n^{\mathrm{F}}(t_0)\lambda_n^{\mathrm{F}} - \frac{\epsilon}{1+\epsilon}\Delta^* Q_n(t_0) \\ &\qquad + Q_n^{\mathrm{R}}(t_0)\lambda_n^{\mathrm{R}} - Q_n^{\mathrm{F}}(t_0)\lambda_n^{\mathrm{F}} \\ &\qquad - \bigg(1 - \frac{\epsilon}{2(1+\epsilon)}\bigg) \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\}\bigg) \\ &\quad + 2\sum_n N\Big(Q_n^{\mathrm{R}}(t_0) + Q_n^{\mathrm{F}}(t_0)\Big) + C_8 \\ &= 2T \sum_n \bigg(Q_n^{\mathrm{L}}(t_0)\lambda_n^{\mathrm{L}} + Q_n^{\mathrm{R}}(t_0)\lambda_n^{\mathrm{R}} \\ &\qquad - \bigg(1 - \frac{\epsilon}{2(1+\epsilon)}\bigg) \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} \\ &\qquad - \frac{\epsilon}{1+\epsilon}\Delta^* Q_n(t_0)\bigg) \\ &\quad + 2\sum_n N\Big(Q_n^{\mathrm{R}}(t_0) + Q_n^{\mathrm{F}}(t_0)\Big) + C_8, \end{aligned}$$

where C_8 is a constant independent of $Z(t_0)$. It is easy to see that

$$\begin{aligned} &Q_n^{\mathrm{L}}(t_0)\lambda_n^{\mathrm{L}} + Q_n^{\mathrm{R}}(t_0)\lambda_n^{\mathrm{R}} \\ &\leq \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\}\bigg(\frac{\lambda_n^{\mathrm{L}}}{\alpha} + \frac{\lambda_n^{\mathrm{R}}}{\gamma}\bigg). \end{aligned}$$

By the constraint (18),

$$\begin{aligned} &Q_n^{\mathrm{L}}(t_0)\lambda_n^{\mathrm{L}} + Q_n^{\mathrm{R}}(t_0)\lambda_n^{\mathrm{R}} \\ &\quad - \bigg(1 - \frac{\epsilon}{2(1+\epsilon)}\bigg) \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\}\bigg) \\ &\leq -\frac{\epsilon}{2(1+\epsilon)} \max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} \end{aligned}$$

Choose large enough T such that

$$\frac{\epsilon T}{2(1+\epsilon)} \geq \frac{2N(N+1)}{\gamma}.$$

Then

$$\begin{aligned} &\sum_n N\Big(Q_n^{\mathrm{R}}(t_0) + Q_n^{\mathrm{F}}(t_0)\Big) \\ &\leq N(N+1)\sum_n \Big(Q_n^{\mathrm{R}}(t_0) + \Delta^* Q_n(t_0)\Big) \\ &\leq \frac{\epsilon T}{2(1+\epsilon)}\bigg(\frac{1}{2}\max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} + \Delta^* Q_n(t_0)\bigg). \end{aligned}$$

Therefore,

$$\begin{aligned}&\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0)\big]\\ &\leq -\frac{\epsilon T}{2(1+\epsilon)}\left(\frac{1}{2}\max\big\{\alpha Q_n^{\mathrm{L}}(t_0), \gamma Q_n^{\mathrm{R}}(t_0)\big\} + \Delta^* Q_n(t_0)\right.\\ &\quad + C_8.\end{aligned}$$

Let

$$\mathcal{B} = \left\{ Z\colon \frac{1}{2}\max\big\{\alpha Q_n^{\mathrm{L}}, \gamma Q_n^{\mathrm{R}}\big\} + \Delta^* Q_n \right.$$
$$\left. \leq \frac{2(1+\epsilon)(\delta + C_8)}{\epsilon T}, \forall n \in [N] \right\}. \quad (28)$$

Then it can be verified that $\mathcal{B}$ is a finite set and

$$\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0) = Z\big] \leq -\delta, \forall Z \in \mathcal{B}^c,$$
$$\mathbb{E}\big[V(Z(t_0+T)) - V(Z(t_0)) \bigm| Z(t_0) = Z\big] \leq C_8, \forall Z \in \mathcal{B},$$

which completes the proof. ■

V. Conclusion

The P2P architecture offers benefits to robustness, scalability, and reduced administrative cost over hierarchical networks. In this paper, we proposed a fully decentralized scheduling algorithm for data-parallel computing systems on P2P networks. The proposed task scheduling algorithm explores a backpressure-based approach, where each node makes decisions on whether to execute some tasks or delegate them, and thus being equal in scheduling. This approach strikes the right balance between data locality and load-balancing in a decentralized fashion, and proves to be throughput optimal, i.e., it maximizes the throughput of the system.

VI. Acknowledgement

This work was supported in part by the NSF under Grant ECCS-1255425.

References

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun.*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] "Hadoop," http://hadoop.apache.org.

[3] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297–1303, Sep. 2010.

[4] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using MapReduce," in *Proc. Ann. ACM SIGMOD Conf.*, Indianapolis, IN, 2010, pp. 495–506.

[5] Y. Ganjisaffar, T. Debeauvais, S. Javanmardi, R. Caruana, and C. V. Lopes, "Distributed tuning of machine learning algorithms using MapReduce clusters," in *Proc. Workshop Large Scale Data Mining: Theory and Applications (LDMTA)*, San Diego, CA, 2011, pp. 2:1–2:8.

[6] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, "SystemML: Declarative machine learning on MapReduce," in *Proc. Int. Conf. Data Engineering (ICDE)*, Hannover, Germany, 2011, pp. 231–242.

[7] S. N. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Comput. Syst.*, vol. 28, no. 1, pp. 184 – 192, Jan. 2012.

[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. USENIX Conf. Operating Systems Design and Implementation (OSDI)*, San Diego, CA, 2008, pp. 29–42.

[9] H. Lin, X. Ma, J. Archuleta, W. Feng, M. Gardner, and Z. Zhang, "MOON: MapReduce on opportunistic environments," in *Proc. ACM Int. Symp. High Performance Distributed Computing (HPDC)*, Chicago, IL, 2010, pp. 95–106.

[10] H. Lin, X. Ma, and W.-C. Feng, "Reliable MapReduce computing on opportunistic resources," *Cluster Computing*, vol. 15, no. 2, pp. 145–161, Jun. 2012.

[11] Y. Ji, L. Tong, T. He, J. Tan, K.-w. Lee, and L. Zhang, "Improving multi-job mapreduce scheduling in an opportunistic environment," in *Proc. IEEE Int. Conf. Cloud Computing (CLOUD)*, Santa Clara, CA, 2013, pp. 9–16.

[12] F. Marozzo, D. Talia, and P. Trunfio, "P2P-MapReduce: Parallel data processing in dynamic cloud environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1382–1402, 2012.

[13] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. ACM Symp. Operating Systems Principles (SOSP)*, Bolton Landing, NY, 2003, pp. 29–43.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE Symp. Mass Storage Systems and Technologies (MSST)*, Incline Villiage, NV, May 2010, pp. 1–10.

[15] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. ACM Symp. Operating Systems Principles (SOSP)*, Big Sky, MT, 2009, pp. 261–276.

[16] T. White, *Hadoop: The definitive guide*. Yahoo Press, 2010.

[17] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. European Conf. Computer Systems (EuroSys)*, Paris, France, 2010, pp. 265–278.

[18] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for mapreduce clusters," in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*, Lisbon, Portugal, 2011, pp. 187–207.

[19] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "BAR: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proc. IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing (CCGRID)*, Newport Beach, CA, 2011, pp. 295–304.

[20] Q. Xie and Y. Lu, "Degree-guided map-reduce task assignment with data locality constraint," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Cambridge, MA, 2012, pp. 985–989.

[21] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*, Turin, Italy, 2013, pp. 1609–1617.

[22] ——, "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*, 2014, to appear.

[23] W. Wang and L. Ying, "Data locality in mapreduce: A network perspective," in *Proc. Annu. Allerton Conf. Communication, Control and Computing*, Monticello, IL, 2014, pp. 1110–1117.

[24] Q. Xie and Y. Lu, "Priority algorithm for near-data scheduling: Throughput and heavy-traffic optimality," in *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*, Hong Kong, China, 2015.

[25] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, pp. 1936–1948, Dec. 1992.

[26] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control and Stochastic Networks Perspective*. New York: Cambridge Univ. Press, 2014.